KeyChains: A Decentralized Public-Key Infrastructure*

Ruggero Morselli Bobby Bhattacharjee Jonathan Katz Michael Marsh
University of Maryland
{ruggero,bobby,jkatz,mmarsh}@cs.umd.edu

Abstract

A Certification Authority (CA) can be used to certify keys and build a public-key infrastructure (PKI) when all users trust the same CA. A decentralized PKI trades off absolute assurance on keys for independence from central control and improved scalability and robustness. The PGP "web of trust" model has been suggested as a decentralized certification system, and has been used with great success for secure email. Although the PGP web of trust model allows anyone to *issue* certificates which can be used to form certificate chains, the *discovery* and *construction* of certificate chains relies on centralized keyservers to store certificates and respond to queries.

In this paper, we design and implement KeyChains, a peer-to-peer system which incorporates a novel lookup mechanism specifically tailored to the task of generating and retrieving certificate chains in completely unstructured networks. By layering our system on top of the web of trust model, we thus obtain the first PKI which is *truly decentralized* in all respects. Our analysis and simulations show that the resulting system is both efficient and secure.

1 Introduction

We present KeyChains, a decentralized public key infrastructure based on the PGP web of trust model. PGP has been a popular way for email users to exchange public keys without resorting to a centralized certification authority (CA). In PGP, users both generate their own keys and certify each other's keys. Users verify a PGP key either by direct certification or using a certificate chain. Such decentralization inevitably trades strong guarantees for weaker assurances. The CA is a trusted third party (TTP), and trust in the CA implies that all keys signed by it are absolutely trusted. In the decentralized case, however, only directly verified keys can be so trusted. The goodness of keys that are certified using a certificate chain depends on how diligent other users are in verifying keys. Hence, decentralized PKIs trade off absolute assurance for greater scalability and independence from centralized control. A CA-based PKI is appropriate for electronic banking and commerce, but the requirement of a single TTP is often too onerous for lower-risk applications such as personal email, posts on Usenet or web forums, instant messaging, etc.

The PGP web of trust model [15, 31, 6] is a notable first step toward realizing a decentralized PKI. However, the PGP web of trust is not itself a PKI, as it does not provide a mechanism for retrieving public keys or certificate chains. In practice, a PKI for PGP is implemented by centralized *keyservers*[1] that store and answer queries about certificates. An obvious extension is to replicate the keyserver data on a set of servers. While this provides fault-tolerance, the number of centralized servers required for a global keyserver cluster must scale linearly with the number of PKI users, and thus would require a significant infrastructure. An

*Technical Report CS-TR-4788, Department of Computer Science, University of Maryland, College Park, MD, 20742, USA. Also filed as UMIACS-2006-12. This work was supported, in part, by: NSF Trusted Computing grant #0310499, NSF-ITR #0426683, NSF grant CNS0426683, NSF CAREER award #0447075, ITR Award CNS-0426683 and DoD contract MDA90402C0428. Bobby Bhattacharjee is also supported by a Sloan Foundation Fellowship.

[1]E.g., `http://pgp.mit.edu`.

ideal solution would be completely decentralized, seamlessly scalable with increasing numbers of users and applications and not present any scope for censorship.

KeyChains is designed in pursuit of this ideal, and builds upon the PGP web of trust. KeyChains is a novel and entirely distributed mechanism allowing users to store their public keys as well as to retrieve the public keys of others. The key location protocol in KeyChains is a modified version of Local Minima Search (LMS) [22]; LMS is an existing object lookup protocol, which provides provable performance guarantees when run over arbitrary unstructured networks. In contrast to previous suggestions for decentralized PKIs [2, 10] based on distributed hash tables (DHTs) [24, 28, 20, 26], KeyChains is specifically tailored to the task of discovering and retrieving *certificate chains* and not just public keys; in fact, our search protocol has the useful property that once a user's public key is located, a certificate chain from the initiator to the target is generated "for free." A second advantage of our approach in comparison to DHTs is that we do not require or impose any structure on the underlying network. These advantages make KeyChains well-suited for implementing a decentralized PKI in systems with large, dynamic user populations and arbitrary trust relationships.

KeyChains incorporates two significant contributions beyond the base LMS protocol and PGP:

- The LMS protocol is designed for undirected topologies, but the PGP web of trust used by KeyChains is directed, and this fact requires a subtle but significant protocol modification (Section 3.1). Further, the base LMS protocol has no concept of trusted links or certificate chains, and KeyChains incorporates these two concepts efficiently into the lookup protocol.
- PGP provides a model and defines a certificate format. KeyChains uses this model to provide a complete PKI system with ability to publish, search and validate keys without relying on central servers.

We also present a detailed evaluation of KeyChains through both simulation and implementation and show that it is efficient, scalable, and fault-tolerant. The remainder of the paper is organized as follows. The next section provides background on the web of trust model, and presents overviews of two possible solutions. Section 3 describes the design of KeyChains in detail, while Section 4 discusses some attacks on the system. Sections 5 and 6 present experimental results from the simulation and implementation, respectively. Related work is discussed in Section 7, and Section 8 summarizes the paper.

2 Distributing the Web of Trust

The PGP web of trust model relies on *certificates* [18]. We denote a certificate by

$$\mathsf{cert}_A(B, PK_B) \stackrel{\text{def}}{=} \langle \text{“}B\text{'s key is } PK_B\text{”}, \sigma_A \rangle ,$$

where σ_A is A's signature (with respect to A's public key PK_A) on the previous statement. We will assign stronger semantics to certificates, and view them as assertions about the *trustworthiness* of B to issue certificates.[2] That is, $\mathsf{cert}_A(B, PK_B)$ represents a signature by A on the statement "B's public key is PK_B and I trust B to issue certificates using this key." We will say certificate $\mathsf{cert}_A(B, PK_B)$ is *issued* by A and *certifies* B. To simplify the exposition, we assume that each (honest) user holds only one public key and therefore sometimes associate a user B with her public key PK_B.

Certificates may be used in the following way: assume a user A already knows the public key PK_B of some user B, and furthermore that A trusts B to issue certificates. Then A can validate C's public key if C can present a certificate $\mathsf{cert}_B(C, PK_C)$ to A. In general, however, it is not likely that C will hold a certificate issued by someone directly trusted by A. The web of trust model therefore introduces *certificate*

[2] In fact, both the web of trust and our system can simultaneously support certificates of both types, and it is only for simplicity of the exposition that we assume the stronger semantics exclusively.

chains. A certificate chain from B to C is a sequence of certificates $\mathsf{cert}_1, \ldots, \mathsf{cert}_\ell$ such that (1) B is the issuer of cert_1; (2) the issuer of cert_{i+1} is certified by cert_i for $1 \leq i < \ell$; and (3) C is certified by cert_ℓ. If A holds PK_B, we may abuse notation and refer to the above as a certificate chain from A to C. Now, A can learn about the public key of C by finding a certificate chain from itself to C; when A searches for such a chain, we speak of A as the "initiator" of the search and C as the "target."

In general, A might want to obtain multiple certificate chains from itself to C and will apply a *trust aggregating function* on the set of constructed chains to determine how much "trust" to place in the binding between C and PK_C. In the easiest example of trust aggregating function, A trusts the binding if a certificate chain exists. A more sophisticated example accepts the binding between C and PK_C if there are at least n disjoint certificate chains of length at most ℓ from A to the binding. See [13, 19, 3, 16] for other examples.

The web of trust model does not specify any particular manner in which certificate chains should be discovered or constructed. The solution in practice has been to rely on central *keyservers* that store certificates. In response to a query about a particular user C, the keyserver returns any stored certificates that certify C. Proceeding iteratively, an initiator can attempt to reconstruct a certificate chain from itself to the desired target C. This is the method currently used to find certificate chains in the web of trust model. The keyservers remain central points of failure and present bottlenecks as the system grows large or the frequency of searches increases.

2.1 A First Attempt: Distributed Hash Tables

The web of trust model is a promising basis for a fully decentralized PKI, because it already distributes the ability to issue certificates. The limiting factor is the lack of an efficient and decentralized mechanism to search for users' public keys and to construct appropriate certificate chains. If a robust, distributed search mechanism could be layered on top of the PGP web of trust, the result would be a truly decentralized PKI.

A natural first attempt is to use a DHT to store and search for public keys. Here, a user B with public key PK_B would insert the index-value pair (B, PK_B) into the DHT; when another user A wants to find the public key associated with B, it would perform a search in the DHT for any data item stored under the index B. This idea has been suggested [2, 10] as an alternative to—rather than an extension of—the web of trust. This approach is vulnerable to a variety of attacks: of particular concern is that there is nothing preventing an adversary from inserting a false public key under the index of another user B. In this case, a search for B's public key would return multiple values and it is unclear how to distinguish a correct key for B from spurious keys. Such an attack is mitigated only if the initiator of a search can authenticate the real public key by, for example, locating a certificate chain. Unfortunately, a DHT-based approach only locates public keys but does not allow efficient reconstruction of certificate chains.

Augmenting the DHT so that it stores certificates in addition to public keys might allow the initiator to reconstruct a certificate chain as in the keyserver model: the initiator would search recursively (in reverse) from the target, hoping to build a certificate chain from itself to the target. Though this will achieve the desired result, it will be incredibly inefficient. Even if optimized, the resulting search may require $O(n)$ searches in a network of size n. The time required might be reduced by caching large parts of the web of trust at each node in the DHT [17, 7]; however, this requires a very large amount of state ($O(\sqrt{n})$ in [17, 7]) and has a high maintenance overhead to keep the cached state current.

2.2 Our Solution

To enable a fully decentralized PKI, we construct an efficient, distributed lookup algorithm called KeyChains that returns *certificate chains* and not just public keys; this can then be layered on top of the PGP web of trust. KeyChains is based on LMS, a peer-to-peer data sharing substrate providing provable performance guarantees in *unstructured* networks [22]. Our key idea is that by taking the PGP "certificate graph" (in which there is an edge from A to B only if $\mathsf{cert}_A(B, PK_B)$ exists) as the underlying peer graph and adapting

LMS to store and search for public keys over this peer graph, we obtain a lookup protocol that returns a certificate chain from the initiator to the target whenever it locates a target public key (this feature relies on the specifics of LMS; further details are given in the next section). It is important to note that KeyChains provides a general mechanism for publishing keys and locating certificate chains, but does not impose any particular trust function for evaluating certificates. Hence, each user is free to choose any particular trust function for evaluating and verifying certificates located by KeyChains.

Adapting LMS to construct the KeyChains application requires a number of modifications to the basic LMS protocol. We stress that the primary advantages of using LMS rather than a DHT are that (1) our approach enables fast lookup of public keys *and certificate chains* in tandem, and (2) LMS can be run over *unstructured* topologies, which is crucial since the underlying topology here (the PGP certificate graph) results from trust associations which cannot be changed to suit the lookup protocol.

3 Implementing a PKI using LMS

We now present our main contribution, the design of a fully decentralized public key infrastructure, KeyChains. This PKI is built on top of the Local Minima Search (LMS) protocol [22], which is capable of efficient storage and retrieval of data over a network defined by a web of trust. We begin with a brief discussion of the LMS protocol and then describe the modifications to this protocol needed to realize our PKI.

A brief overview of LMS is given in Appendix A. Here we provide only the general concepts and terminology. The semantics of LMS are similar to those of a DHT: peers and objects are mapped into an *identifier space* using consistent hashing, and objects are stored at peers determined by the distance between objects' and peers' identifiers in this space. Each peer knows the identifiers of peers within h hops of it in the network, which defines its h*-hop neighborhood.* Rather than storing an object at the peer with the globally smallest distance between their identifiers, LMS stores multiple *replicas* of the object at *local minima*: peers with the smallest identifier distance in their neighborhoods.

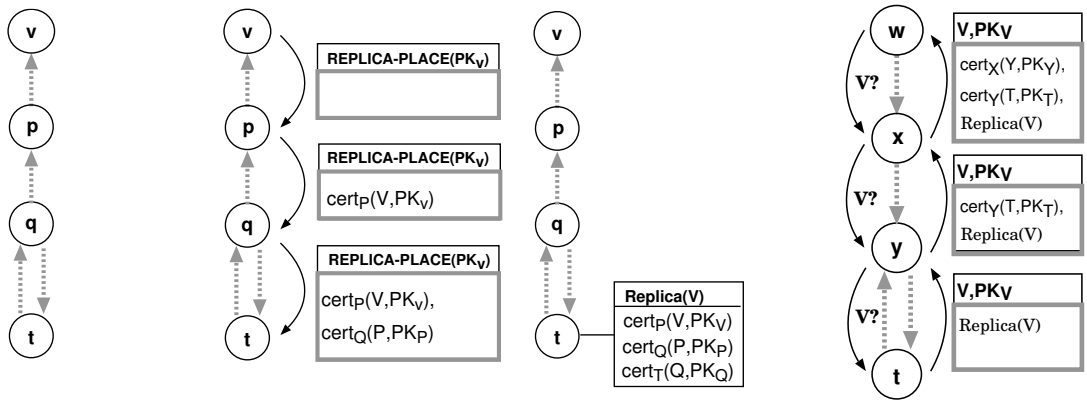
A peer performing either storage or lookup sends a number of *probes* into the network. These probes are forwarded to local minimum using first a fixed-length random-walk (mixing) phase followed by a deterministic phase. Forwarding is always done along undirected links between peers; peers never contact one another directly except according to the overlay graph. Local minima are selected randomly, and the performance of the protocol relies on storing enough replicas and performing enough searches that there is a high probability of locating at least one replica.

3.1 KeyChains

The properties of LMS are nearly ideal for implementing a PKI. First, because LMS runs over arbitrary topologies it can be run on a peer-to-peer system where the topology reproduces the web of trust "certificate graph." Furthermore, due to the way probe messages are forwarded in LMS, using LMS to search for public keys means that whenever a probe locates a target public key, a certificate chain from the initiator to this target can be reconstructed directly from the path taken by the probe. Actually creating this certificate chain requires significant modifications to LMS, including extending the protocol to directed overlay graphs.

We assume that each *principal* (user) is associated with a *peer*, a host that runs the KeyChains protocol. KeyChains does not impose specific trust relationships between peers and principals. Multiple principals may be associated with a single peer; e.g., a departmental server can hold certificates for all department members. In the rest of the discussion, however, for simplicity we assume that there is a one-to-one mapping between peers and principals. Peers never need to know the private keys of their principals, so compromising a peer does not compromise its principal.

Principals generate certificates for one another using some out-of-band mechanism, and relay these new certificates to their peers. Recall that a certificate from A to B, denoted by $\mathsf{cert}_A(B, PK_B)$, is the



(a) Initial KeyChains network. Validated links shown.

(b) Placement messages.

(c) Replica stored.

(d) Replica search.

Figure 1: Replica placement and search in KeyChains. The `REPLICA-PLACE` message for key PK_V is sent from initiator v along reverse validated edges, collecting the certificates corresponding to the edges on the path. Finally, the message arrives at local minimum t, who stores PK_V together with the certificate path. Search involves forwarding a message along validated edges until it reaches the local minimum t, which then responds with the stored replica and constructs the certificate path back to w.

pair $\langle$ "B's public key is PK_B and B is trustworthy", $\sigma_A \rangle$, where σ_A denotes A's signature on the statement. The PGP web of trust is simply the union of all certificates existing in the system. Principals can associate a measure of trust (that is, how likely a principal is to issue *correct* certificates) with a certificate, and a final trustworthiness can then be computed from a certificate chain [19]. PGP allows for different levels of trust, though these values are not exported to the keyservers.

Principals and peers are distinct, each having its own public key. This means that when principals exchange public keys, they must also exchange the public keys of their peers. This allows one peer to authenticate another when they connect, and we say that this in turn *validates* the link between the principals. The set of all validated links defines the *trust graph*. A directed edge from A to B exists in this graph if and only if there exists a certificate $\mathsf{cert}_A(B, PK_B)$ and A's peer can authenticate its connection to B's peer.

3.1.1 Adapting LMS to the Trust Graph

We express trust with certificates that have an inherent directionality, so using these certificates requires changing the undirected (that is, bidirectional) links of LMS to directed links. This invalidates the analysis of LMS [22], but in practice we expect the results to hold approximately due to the large number of links that likely *will* be bidirectional. For the PGP graph [29], about $2/3$ of the links in the large strongly connected component are bidirectional.

We observe that each certificate in the web of trust implies a directed link between peers, so each certificate chain implies a corresponding (directed) path. When forwarding messages, links that "point" in a particular direction should be used, and the corresponding certificates added to the chain. The appropriate directionality for links is somewhat subtle, so we delay the discussion until we present the details of the PKI operations. We note, however, that we must redefine the h*-hop neighborhood* of a peer v as the set of all peers to which there exists a *bidirectional* path from v with length at most h. This redefinition implies that the path taken during deterministic forwarding is reversible: we can create a certificate chain in either direction.

3.1.2 Placing Replicas of Public Keys

Suppose v (Fig. 1(a)) is the peer for principal V (we use this capitalization convention throughout). A PKI `store` operation invoked by V involves sending a `REPLICA-PLACE` probe for each replica that v needs to place. When a local minimum receives a duplicate storage request, it rejects the request and notifies v. For each failed probe v doubles the length of the random walk and sends a new probe, until the number of failures exceeds some configured threshold.

As a probe from v is forwarded (Fig. 1(b)), the certificates corresponding to traversed edges are appended to the message, thus constructing a certificate chain. Since we want this certificate chain to point toward V, probes are forwarded along *incoming* links; that is, a peer p forwards a probe to a peer q only if the certificate $\mathsf{cert}_Q(P, PK_P)$ exists in the trust graph. When the probe finally reaches a local minimum t, a replica containing the public key of V and the constructed certificate path is stored at t (Fig. 1(c)). Note that the certificate chain being stored is directed from T to V.

3.1.3 Finding Replicas of a Public Key

Suppose W wants to locate V's public key. A PKI `retrieve` operation invoked by w involves sending `SEARCH-PROBE` messages for the identifier of V (Fig. 1(d)). As the `SEARCH-PROBE` message is forwarded, certificates corresponding to traversed edges will again be appended to the message. Here, we want this certificate chain to point *from* W, and therefore search probes are forwarded along *outgoing* links (i.e., a peer p forwards a probe to a peer q only if the certificate $\mathsf{cert}_P(Q, PK_Q)$ exists in the trust graph). When a peer t receives a probe for a key it holds, it may respond with the value of the key. Note that any such t also holds a certificate chain from T to V (stored during placement of V's key). Furthermore, the `SEARCH-PROBE` message at this point contains a certificate chain from the initiator W to T. By "gluing" these certificate chains together (specifically, by having the `SEARCH-PROBE` message retrace the path back to the source, extending the certificate chain at each hop; see Fig. 1(d)), we obtain a certificate chain from W to V as desired. We note that because deterministic forwarding to a local minimum (following the random walk phase of a probe) is done along bidirectional links, a local minimum for a search is guaranteed to be a local minimum for a placement.

If the search probe reaches a minimum that does not contain V's replica, w sends a new probe (and w repeats this up to some configured maximum number of times s). If multiple probes reach the same local minimum without finding a replica of the key, w doubles the length of the random walk used in subsequent probes.

3.1.4 Revoking a Public Key

From time to time, a previously published public key will need to be invalidated, generally when the corresponding private key has been exposed. This problem is easily solved in KeyChains, taking inspiration from existing techniques: certificate expiration, revocation lists and online certification. Expiration times can easily be incorporated into the certificates stored by KeyChains (X.509 and PGP certificates both support them). Explicit revocation in KeyChains can be accomplished with a two-pronged approach. The peer that publishes a public key keeps track of the peers that store a replica of that key; upon revocation, the peer requests that all such replicas be deleted, so that new searches will not find the revoked key. At the same time, the peer places revocation statements for the key at a number of local minima, so that a search for the key is likely to return a statement. Note that a malicious peer can store (and serve) a revoked key, much like a malicious user can cache and use revoked keys in any PKI. However, if a user periodically searches for revocation statements of previously retrieved keys, he will promptly find out if a key has been revoked.

4 Attacks on the PKI

In this section, we argue that KeyChains has very strong security properties: the system is resilient to a wide range of attacks. We describe potential attacks in terms of an adversary $\mathcal{A}$.

A particularly powerful adversary is one that corrupts principals and is therefore able to insert itself into the trust graph and create certificates with whatever public keys it chooses bound to any principal (in the network or not). Note that such an adversary is, in effect, attacking the web of trust model rather than our particular system. The decentralized nature of the PKI, however, mitigates the damage that even such an adversary can cause. Peers accept bogus certificates only if they are unable to find more trustworthy correct certificates, so only the peers in the immediate neighborhood of corrupted principals are expected to be significantly impacted. Because this is a more general attack against the model (and is, in fact, inevitable in the absence of a centralized authority), rather than present the details here we defer them to Appendix B.

For the attacks against the protocol, we first restrict $\mathcal{A}$ to actively interfering with communication between peers, but not corrupting any peers. We need only consider message deletion attacks, since message insertion and modification attacks are easily dealt with using well-known techniques (recall that messages are routed only between peers who share a secure channel). In a message deletion attack, $\mathcal{A}$ targets some of the peers and deletes most or all of the messages to and from them, effectively cutting them off from the rest of the network. If $\mathcal{A}$ can cause specific subsets of nodes to fail, it can partition the network. We simulate this attack in Section 5.2, and show that KeyChains provides resilience against different forms of this attack.

A more powerful adversary might be able to corrupt peers. The adversary is unable to generate new certificates, since it does not know any principals' private keys. It can, however, cause the corrupted peers to either refuse to participate or attempt to bias the behavior of the protocol. Refusal to participate is the same as the previously discussed attack. Biasing the protocol behavior can take the form of either a denial of service attack or a *path-biasing* attack on the certificate chains returned during successful public key retrievals.

In the denial of service attack, the corrupted peers assert that any placement probe forwarded to them has succeeded even though no replica is stored and the remaining path from the corrupt peer might be entirely bogus. These peers then silently drop any search probes they receive. This attack, and the PKI's resilience to it, is discussed in Section 5.2.

In the path-biasing attack, $\mathcal{A}$ returns valid certificate chains distributed differently from the certificate chains that KeyChains would naturally return. For instance, $\mathcal{A}$ may cause only "long" certificate chains to be returned with the assumption being that users will view such chains as untrustworthy. In order for this attack to be effective in a large network, $\mathcal{A}$ has to compromise a sizable fraction of the peers. This is because (in an undirected network), if the adversary controls only a small fraction of the peers, the probability that any individual probe of an uncorrupted peer v is affected is very small for all but a small fraction of the honest nodes [22].

5 Experimental Results

In this section we present a study of the performance of KeyChains, including a comparison with distributed PKIs built using other lookup techniques. We also evaluate the performance of KeyChains in the presence of failures and its scalability. Our results highlight the advantages of KeyChains in this problem domain. The data presented for the implementation (Section 6) provide additional context useful in evaluating the protocol costs presented in this section.

We use as the trust graph the PGP keys and certificates provided by the `keyanalyze` project [29], of which we consider only the largest strongly connected component (containing 25584 keys). On top of this graph we run our message-level protocol simulator, equating each key in the graph with a unique KeyChains principal. Identifiers are assigned randomly to principals to form the peer-to-peer network. Thirty such sets of assignments are made, each effectively generating a distinct randomized trust network.

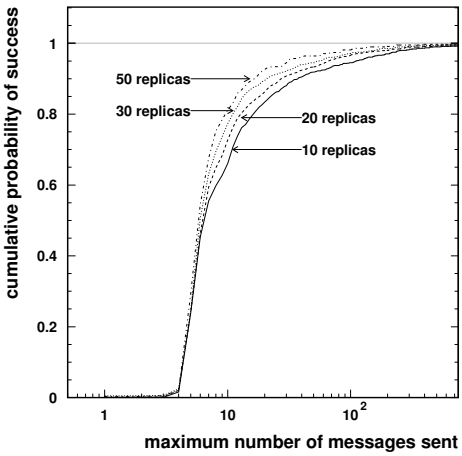


Figure 2: Probability of success vs. number of messages sent in KeyChains, for different numbers of replicas. Note that the number of messages is given on a logarithmic scale.

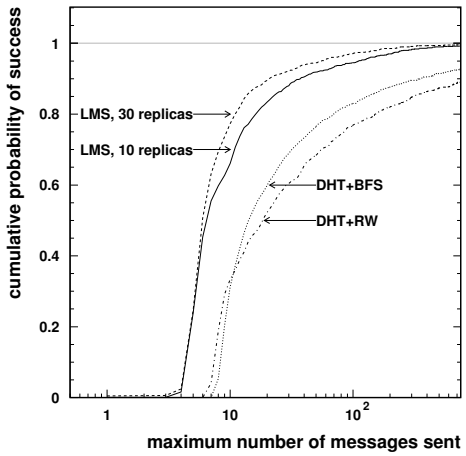


Figure 3: Probability of success vs. number of messages sent in different designs of a decentralized PKI. KeyChains (LMS) performs noticeably better than the DHT-based approaches.

An experiment involves randomly selecting twenty pairs of peers. For each pair, the first peer places a number of replicas (denoted by r in Section A) after which the second peer attempts to locate one or more of the replicas by generating search probes one at a time, until it succeeds. After 150 unsuccessful probes an operation is deemed to have failed. One such experiment is performed for each of the thirty trust networks and desired number of replicas, so for each value of r we obtain 600 data points.

The actual probes have an initial random walk length of 3; the maximum permitted random walk length is 25. Peers with comparatively few neighbors are more likely to encounter failures, so as an optimization we allow these peers to use more highly connected neighbors as *proxies* for initiating random walks both for replica placement and search.

The performance of searches in KeyChains is shown in Fig. 2 as the probability of success when a given number of messages have been sent. Message totals include all messages exchanged over all probes used in a particular search, and so aggregate the transmission costs of all the probes. As expected (see Section A), as the number of replicas in the system increases, fewer messages need to be exchanged in order to find one of them.

In Table 1 we show how adding replicas improves the search performance of KeyChains. For a given probability of success and level of replication, we give the average number of messages exchanged in a successful search and the largest number of messages exchanged in any of those searches. The latter, which are used to produce Fig. 2, are a conservative measure of the protocol performance, since they probe the tails of the message cost distributions.

With only 10 replicas, when 99% of the searches succeed, the average number of messages sent by all probes in a search is only 25, which is reasonably low. The distribution of message exchanges has a very long tail, however, with the poorest-performing search requiring 462 messages. Increasing the number r of replicas brings the maximum and average down rapidly. As r increases, both the average and maximum decrease as approximately $1/\sqrt{r}$.

	10 Replicas		20 Replicas		30 Replicas		40 Replicas		50 Replicas	
Pr[success]	**Avg**	**Max**	**Avg**	**Max**	**Avg**	**Max**	**Avg**	**Max**	**Avg**	**Max**
90%	12.1	38	10.2	28	8.86	22	8.05	18	7.97	18
95%	17.1	112	13	68	11	54	9.83	46	8.91	32
99%	25	462	19	362	15.3	244	17.6	450	11.4	136

Table 1: Number of Search Messages Sent for KeyChains

5.1 Comparison with DHT-Based Systems

One might imagine storing public keys in a distributed hash table. The basic problem with this approach is that a DHT provides no assurance that the key returned actually maps to the desired principal (see Section 2.1), unless the DHT can be constructed such that a peer p's routing table contains entries only for peers that p trusts. Since this is not generally possible,[3] a DHT-based design needs an additional mechanism to retrieve certificate chains after the initial key lookup. Placement and lookup protocols for DHTs are described elsewhere [28, 26]; we merely use the result from [28] that the average number of messages needed to place or find an item is $\frac{1}{2}\log n$.

We simulate two possible augmented DHT designs, corresponding to sensible approaches to certificate-chain discovery after the initial key lookup. The first (DHT+BFS) performs a breadth-first search on the trust graph for the shortest-path certificate chain between the search initiator and the target. The second (DHT+RW) performs a random walk on the trust graph from the initiator to the target. To ensure that our comparison with KeyChains is fair, each peer in the DHT knows its 2-hop neighborhood in the trust graph as well as how to contact the peer of every principal for which it holds a certificate (obviating the need for additional DHT lookups). Such neighborhood information is used to make the certificate chain construction more efficient. All design decisions and simplifications are biased toward improved performance for the DHT systems. Only one replica is placed in the DHT systems, because we expect that additional replicas will not noticeably improve the performance of certificate-chain construction (though they may slightly improve the efficiency of the initial search for the key).

Fig. 3 compares these two DHT designs to KeyChains; each of the DHT experiments comprises at least 1200 placements and searches. The most striking feature of the data is that the number of messages required by KeyChains to achieve a particular probability of success is always noticeably lower than the number of messages required by either DHT-based system. With 10 replicas KeyChains achieves a 90% probability of success with no more than 38 messages, while DHT+BFS requires as many as 358 messages. The difference in average numbers of messages needed is also substantial: 12.1 for KeyChains and 66.7 for DHT+BFS. Of the two DHT designs, the breadth-first search outperforms the random walk. We conclude from this comparison that distributed hash tables, while very efficient for data storage and retrieval, are ill-suited for constructing certificate chains in a network with unstructured trust relationships.

Cost of placement. Placing a single DHT item is less costly than placing replicas in KeyChains, so we must additionally consider this cost (again measured as the number of messages exchanged) as a part of our comparison. Table 2 shows the average number of messages needed in KeyChains for replica placement. The per-placement cost increases with extra replicas since more probes return duplicates. While these costs are high, we note that they are amortized over all searches for the replicated key. In particular, with 10 replicas and a 90% probability of success, after 11 searches for a key (on average) KeyChains has a lower total cost than DHT+BFS. If we instead use the maximum search costs, only 2 searches are required to make up the difference in placement costs.

Path length and stretch. On average, the certificate chains returned by KeyChains are twice as long the shortest paths (which are found by DHT+BFS). We have experimented with a slightly modified version of

[3] We discuss this for one particular system [21] in Section 7.

	Number of replicas			
	10	20	30	40
Pl. cost	586	1552	2605	3797

Table 2: Cost of placement in KeyChains

the KeyChains protocol that can be used to find certificate chains that are, with high probability, the shortest. The protocol modification involves carrying neighborhood state in the search probes, and consequently increases the message size requirements of KeyChains; in particular, each probe should carry with it the neighborhoods of all the nodes on the path. If a typical neighborhood contains 100 peers and a typical path length is 20, this increases the size of a message by (20 bytes $\times$ 100 $\times$ 20) = 40 kbytes. If an application depends critically on short paths, this cost might be acceptable.

5.2 Node Failures

Robustness is one of the motivating features for a decentralized PKI; in this section, we examine the behavior of KeyChains in the presence of (potentially adversarial) failures. We consider first the "fail-stop" model: when peers leave the system (for whatever reason) their neighbors are able to detect their departure. We then turn to compromised peers that might exhibit Byzantine (arbitrary) behavior. Compromised peers are limited only in that they do not have access to their owners' private keys, and hence cannot produce incorrect certificates.

Fail-stop failures The risk with failures in the fail-stop model is that the network might become partitioned. While this prevents peers in different partitions from finding each other's newly placed keys, replicas placed before the partition might still be found in either partition. Because the failure of a peer does not invalidate its principal's certificates, certificate chains held with replicas are still valid, and can establish a chain of trust between the search initiator and the target even if they cannot reach one another through the network. Specifically, we find that if a fraction f of randomly chosen peers fail, then slightly more than $1 - f$ of the public keys remain findable from a peer chosen randomly from among the survivors.

An interesting measure of the system with these failures is the probability of success, given the existence of a path of non-failed nodes from the searcher to a local minimum that holds a replica of the key (the probability of finding findable keys). Fig. 4 shows this success probability as a function of messages exchanged when 20 replicas of each key are placed before the failures. Each experiment removes a fixed fraction of the peers, and lost replicas are not replaced. We see from these results that KeyChains degrades rather gracefully. With up to 5% of the peers failing, the loss of functionality is minimal (aside from the unavoidable loss of a fraction f of the replicas), though by the time that 40% of the peers have failed there is substantial degradation. It is difficult to imagine any system performing well under these circumstances. Note that we would expect additional replicas, placed after the failures, to greatly improve the likelihood of finding certificate chains and reduce the cost of finding them. Overall, this is an encouraging result that demonstrates the resilience of the underlying randomized algorithmic techniques used by LMS and KeyChains.

Adversarial failures The goals of an adversarial attack on a PKI are to inject an incorrect public key for a principal or to disrupt the functioning of the system. Clearly, incorrect keys cannot be injected unless the adversary is able to corrupt principals and obtain their private keys. (In this case, the attacks reduces to an attack on the underlying web of trust rather than our specific placement-and-search mechanism. Standard defenses against this attack, such as retrieving multiple certificate chains can be used to mitigate such attacks.)

The most interesting attack is the attempted disruption of the system. It is less powerful than the bogus-key attack above, and we expect KeyChains to exhibit some resilience to it. The attack is simple: a com-

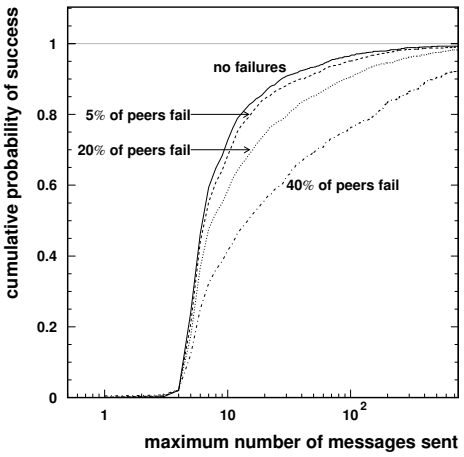


Figure 4: Probability of success vs. number of messages sent for findable public keys. 20 replicas are placed for each public key.

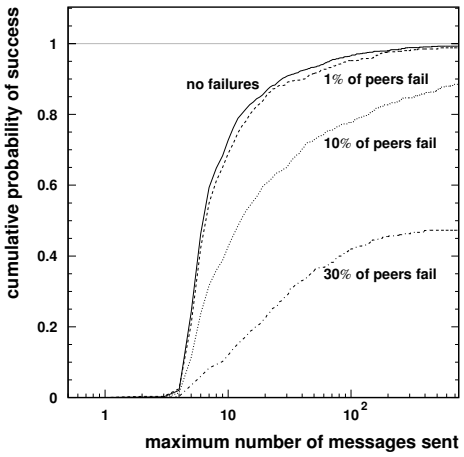


Figure 5: Probability of success vs. number of messages sent in a system under attack. 20 replicas are placed for each public key.

promised peer always reports a bogus success response when it receives a replica placement message, and it silently drops all search probes. Fig. 5 shows the performance of KeyChains under this attack when 20 replicas are "placed" and a fixed fraction of the peers (chosen uniformly at random) are compromised. While the performance of the system degrades rapidly with the number of failed nodes, it should be noted that a 75% probability of success in 100 messages or fewer is remarkable with 10% of the nodes exhibiting this malicious behavior. Even at 30% node failure the system still provides some functionality, albeit greatly reduced.

We now consider the case in which an adversary can compromise a selected set of nodes in the web of trust. In a PGP-like graph, a large number of paths traverse the highest degree nodes, and in Figure 6 we plot the success probability versus message cost when these highest degree nodes in the graph are compromised. As expected, cost increases dramatically, but KeyChains is still able to provide some service. With the top 10 nodes in the graph compromised, KeyChains still performs well, though noticeably worse than in the absence of the attack. With 50 or more of the highly connected peers compromised, the performance drops beyond what most users would find useful, but it is unlikely that any protocol based on the web of trust will be able to perform well in this scenario.

5.3 Scalability

A fundamental premise of our work is that the efficiency of KeyChains will scale well as the web of trust grows. We only have access to one real trust graph of any appreciable size, so in order to test the scalability of KeyChains we generate a number of large graphs having similar characteristics to the actual PGP graph. Modeling the web of trust graph is a very hard problem (see, for example, Capkun et. al. [8]), and is beyond the scope of the present work. Instead, we choose to capture a few basic structural properties of the web of trust graph. In particular, our synthetic graphs replicate the power-law structure of the PGP graph. We give a more detailed description of our model in Appendix C.

In Fig. 7, we plot the performance of KeyChains over a set of power-law graphs generated via our method. For all graphs we place 20 replicas of each key. KeyChains scales seamlessly, incurring little penalty going from a 10,000 node graph to a 100,000 node graph. Obviously, our model is not a faithful reproduction of the PGP web of trust, but it is clear that KeyChains scales well in power-law graphs. We

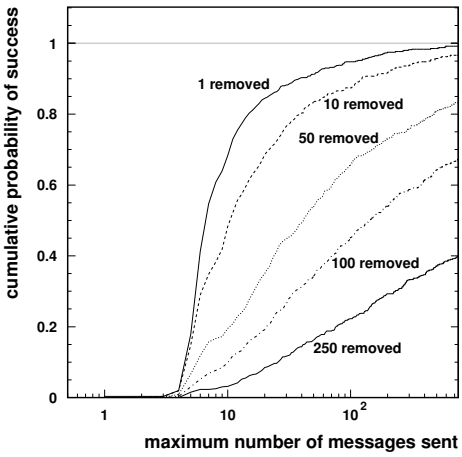


Figure 6: Probability of success vs. number of messages sent when high-degree peers are under attack. The different lines correspond to different numbers of peers removed, where the highest-degree node is removed first, then the next-highest, and so on.

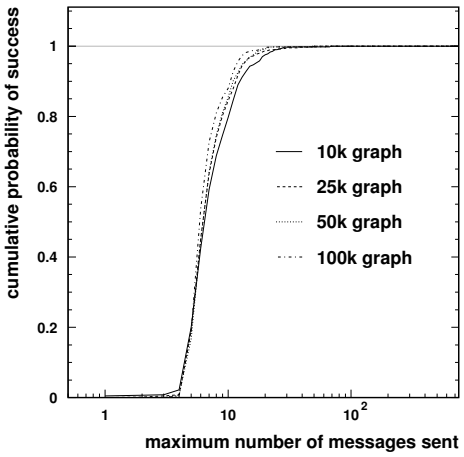


Figure 7: Probability of success vs. number of messages sent for power-law graphs of varying sizes. For all graphs we place 20 replicas of each public key.

believe the reason for this is twofold: power-law graphs of a given size have a relatively small diameter, which allows KeyChains to use short random walks to reach almost any node in the graph. Also, the (very) high degree nodes in large power-law graphs mean that a few local minima will attract deterministically forwarded probes from a large fraction of the network. As long as there are replicas at these network-dominating minima, the search is very efficient.

6 Implementation Benchmarks

In this section we present microbenchmark results from our prototype implementation of KeyChains. Our results show that KeyChains can be deployed in large networks without undue processing, storage, or bandwidth overheads. We show that messages in KeyChains do not grow too large, the state stored by an individual peer is small, and message processing times are not unreasonable. Processing is (heavily) dominated by cryptographic signature generation, though signatures are only generated for valid messages that are to be forwarded or for responses. We do not examine end-to-end latency, since this depends critically on actual paths taken in the underlying network, while our (small) test network runs over the loopback interface on a single host.

The PKI implementation is split into two separate programs: the peer and the user interface client. Multiple clients can connect to a single peer, with different users, though one user is identified as the *owner* of the peer. The peer maintains an access control list indicating what operations (key storage, key retrieval, and peer management) are permitted to a particular user. Each peer and user is associated with a unique 160-bit identifier. A peer's identifier is the SHA-1 hash of its 1024-bit RSA public key. A user's identifier is the SHA-1 hash of his or her email address, as embedded in an X.509 certificate also containing the user's public key.

For each of its neighbors, a peer holds the host and port number at which the neighbor can be reached, public keys and identifiers for the neighbor and its owner, trust statements signed by the peer's and neighbor's owners, and a flag indicating if the neighbor is believed to be online. This information is provided by the

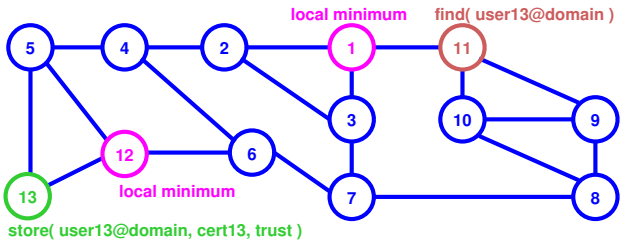


Figure 8: The network used in testing the implementation. All links are symmetric and have the same trust value, and $h = 2$. One particular test is shown, where peer number 13 places its owner's certificate and peer number 11 searches for it. Peers 1 and 12 are the local minima for the certificate.

Message Type	Base Size	Incremental Size
Nbrhood Update	261 B	53 B/neighbor
PK Search	583 B	320 B/path entry
PK Replication	1160 B	same as for search

Table 3: Average Message Sizes

owner through the client interface, though a peer might also read neighbor information from a file. For its h-hop neighbors (including itself), a peer holds the neighbor's identifier, the identifier of its next hop toward that neighbor, the number of hops to the neighbor, and flags indicating whether the neighbor is reachable through outgoing trust links or incoming trust links. In addition, the implementation supports non-binary trust values expressed as integers between 0 and 100, inclusive.

The source code for the implementation is written in C++ (2.6K lines of code) and is based on the publicly available LMS library and on publicly available cryptography [23] and distributed systems [9] software packages. Peers are multi-threaded, and communicate using UDP; clients are single-threaded, and connect with peers using TCP.

In order to understand the quantities affecting the scalability of the system, we test the implementation on a small network of thirteen peers, all running on a single host with two 2.4 GHz Pentium IV processors and running Red Hat Enterprise Linux. The network is shown in Figure 8. After the peers exchange neighborhood information, ten separate experiments are run and their results aggregated. In each experiment, one of the peers stores a single replica of its owner's public key and another peer searches for the key. No additional protocol state or traffic is present during an experiment.

Protocol messages A major factor in the scalability of the system is the size of the messages exchanged. Large messages consume the available bandwidth without allowing sufficient requests to be processed in parallel. We examine three types of messages: neighborhood update, public key search, and public key replication, and the message sizes are summarized in Table 3. All messages have a variation in size of a few bytes due to ASN.1 serialization.

We begin with the neighborhood updates that precede the store-and-retrieve experiments. Twelve of the thirteen peers in the network have three neighbors, and the last has only two. Three-neighbor neighborhood updates have an average size of 420 bytes, with an RMS deviation of 4 bytes, while the two-neighbor update is 367 bytes. Because the updates only include 1-hop neighbors, the difference between these values gives us the incremental cost of each neighbor included in an update, or 53 bytes. Given the incremental cost of each neighbor, we further determine that the average base size of a neighborhood update message is 261 bytes, much of which is taken up by the sender's digital signature. Most updates will only need to propagate changes to the topology, and hence can be made considerably smaller.

Search probes, when initially sent, have an average size of 583 bytes. This includes one trust path element, but because search probes will always contain at least one path element we include this in the base

message size. The initial size of a replica placement message is 1160 bytes. The difference between these, 577 bytes, is the size of a serialized public key certificate. Paths increase and decrease (due to pruning out redundant hops) in length as peers forward a probe. The sizes of these changes are measured to be in multiples of 320 bytes.

Responses to probes include the (pruned) path taken from the sender to the local minimum, as well as the path taken as the response is forwarded back to the sender. This makes response sizes more difficult to analyze, since they will depend heavily on the particular network. In addition, replicas are stored as the entire (signed) replica placement probe that was received by the local minimum, which includes the path from the placing peer to the local minimum.

The largest messages will typically be responses to search requests, which include three paths: the path from the placing peer to the local minimum, the path from the searching peer to the local minimum, and the path from the local minimum back to the searching peer. While our prototype uses UDP for inter-peer communications, we expect that a wide-area deployment would use TCP for reliability and congestion control, and hence could better handle larger messages.

Peer state Another significant measure of scalability is the amount of state a peer must store. Object sizes are difficult to measure when pointers are used, so we instead examine the increase in the heap size when new state is added to a peer. The heap grows by approximately twice the size of the serialized data, due to redundancies, overhead, and bookkeeping. This means that a 2.7 KB serialized replica (again assuming a path length of 5) will take approximately 5.4 KB in memory. Each immediate neighbor is represented in about 0.7 KB of serialized data, and hence contributes about 1.4 KB of state data. h-hop neighbors require about 0.1 KB. The numbers show us that the memory requirements for storing neighborhood information are almost negligible. Similarly, public key replicas are relatively small, with an inexpensive 40 GB drive able to store millions of individual replicas.

Processing benchmarks The final issue we address is processing time. The processing time for a message is taken as the time between when a peer receives a message and when it forwards the next message, as reported by `gettimeofday`. This time averages 30.0 ± 0.4 ms, with a slight increase of 0.2 ms/KB as message size increases, or a processing capacity of over 30 messages per second (assuming only one processor is available to the peer). We note that the processing time for a single message is essentially independent of the overall size of the network.

We expect that processing time is dominated by message signing, since cryptographic operations tend to be expensive. Because the processing time for a message is very short, profiling a running system is not practical. Instead, we perform 1000 signature generations, and measure the total time taken. From this, the average time to sign a message is 13.86 ± 0.01 ms. Two signatures are generated for each message, one for the data that is added to the path and one for the message as a whole, so message signing takes 27.72 ± 0.02 ms, or about 92% of the processing time.

7 Related Work

Resilience has been added to Certification Authorities in Ω [25] and COCA [30]. Both protect data from Byzantine faulty servers using threshold cryptography, though with different approaches and synchrony models, and both are online CAs and hence provide low latency, high availability issuance and revocation of certificates. These systems are still, however, centralized, limiting their potential scalability, but the certificates they return are authoritative, while those returned by KeyChains are only probabilistically correct.

A more scalable solution is presented in RFC 2535 [1], which extends DNS with a `KEY` resource record that can be used to retrieve a public key associated with a domain name. This can be used to construct a more general-purpose PKI [14], though trust devolves to DNS zone administrators, which provides less authoritativeness than fully centralized designs but slightly more than our system. Entry into this system is

still more heavily controlled than in KeyChains, and there are more easily isolated points of failure, limiting the resilience of the scheme.

PGP [15, 31, 6] is the best-known example of a web of trust certification scheme. PGP uses public key cryptography to provide privacy and authentication for email. In PGP, a user can *sign* another user's (public) key, i.e. he produces a certificate binding the identity of the other user to his key; unlike our web of trust model (see Section 2), such a certificate does *not* imply that the issuer trusts the subject. A PGP user also has a private database in which he assigns a trust value to other users whom he knows directly; a certificate chain is useful only if all intermediate users in the chain appear in the database. Certificates in SPKI/SDSI [12, 11] and KeyNote [5] use the concept of *local names*, i.e. names that are unique and meaningful only within a certain name space; a name space is defined by a public key. That is, Alice does not issue a certificate for Bob, she issues a certificate for "Alice's Bob." This certificate is valid to anyone who (transitively) knows Alice. These certificates are also more general; in addition to certifying *principals*, they might also express *authorizations*. It is important to note that these certification systems (including PGP) define data formats, not actual PKIs. For PGP, a PKI is implemented by storing certificates on a well-known set of keyservers.

ConChord [4] implements a PKI for SPKI/SDSI certificates, storing certificates in a Chord [28] ring of peers. For each certificate, ConChord also stores all irreducible derived certificates, which they call "closures." These closures allow for very efficient group membership testing. In general, these closures do not facilitate the construction of certificate chains, and hence ConChord does not provide assurance for the certificates it returns. Assurance could be added by composing all trust relations as part of the closure, though this would greatly increase the storage requirements placed on the system as well as the computational load when closures must be recomputed.

A technique for constructing a decentralized PKI for mobile ad-hoc networks is presented in [7]. Here, each node stores a subset of the web of trust graph such that with high probability, any pair of nodes can construct a certificate chain to each other using only their locally stored certificates. As we mentioned earlier, it is not clear how this scheme would scale since nodes need to store a current view of $O(\sqrt{n})$ certificates. In contrast, KeyChains needs to know the identities of nodes within h hops in the web of trust, but only needs to store certificates for immediate neighbors. The functionalities of KeyChains and the system in [7] differ, as well: KeyChains allows a user to obtain the public key for an identity, while the system in [7] allows a user to verify that a known public key is bound to a particular identity. In [2, 10], a decentralized key store is built using a DHT that provides efficient lookup with high availability. The protocol does not, however, provide any assurance that key bindings are correct.

SPROUT [21] adds trust edges to Chord [28] in order to enable trusted message forwarding. As in KeyChains, 2-hop information is stored, though in SPROUT this is used to find more productive trusted paths (that is, paths that route a request closer to the target). While this is an improvement over a standard DHT in terms of trust, in order to guarantee that the system forms a DHT SPROUT nodes must still employ untrusted links, and thus SPROUT does not form a suitable basis for a system to efficiently construct certificate chains.

8 Summary

In this paper we have presented the design of KeyChains, a decentralized public key infrastructure based on a PGP-like web of trust model. The system uses LMS [22], a peer-to-peer lookup protocol that is efficient for unstructured network topologies. Our protocol has the advantage that searches return not only the public key bound to a particular principal, but also return a complete certificate chain assuring the initiator of the correctness of the key.

We have compared the performance of KeyChains to other possible decentralized solutions; our results show that the integrated key and certificate lookup mechanism in KeyChains makes it more efficient than existing schemes for any desired probability of success.

We have shown empirically that KeyChains is robust against both fail-stop and certain adversarial fail-

ures. In addition, we have discussed the security of the system more generally, for adversaries ranging from active wiretappers to malicious trusted principals. While for the latter we cannot provide strong analytic guarantees (and indeed neither can *any* system based on a web of trust), we argue that KeyChains will, in general, be able to provide correct and trustworthy public key bindings for most principals.

The current trend in distributed systems is toward decentralization. KeyChains is an illustrative example of a decentralized trust-based application; we expect more such applications to emerge in the near future.

References

[1] Donald E. Eastlake 3rd. Domain name system security extensions, March 1999. RFC 2535.

[2] K. Aberer, A. Datta, and M. Hauswirth. A decentralized public-key infrastructure for customer-to-customer e-commerce. *International Journal of Business Process Integration and Management*, 1(1):26–33, 2005.

[3] Advogato's trust metric. `http://www.advogato.org/trust-metric.html`.

[4] Sameer Ajmani, Dwaine E. Clarke, Chuang-Hue Moh, and Steven Richman. ConChord: Cooperative SDSI certificate storage and name resolution. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 141–154, London, UK, 2002. Springer-Verlag.

[5] Matt Blaze, Joan Feigenbaum, and Angelos D. Keromytis. KeyNote: Trust management for public-key infrastructures (position paper). *Lecture Notes in Computer Science*, 1550:59–63, 1999.

[6] J. Callas, L. Donnerhacke, H. Finney, and R. Thaye. OpenPGP message format, May 1996. RFC 1951.

[7] S. Capkun, L. Buttyan, and J. P. Hubaux. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, page 17.

[8] S. Capkun, L. Buttyan, and J. P. Hubaux. Small worlds in security systems: an analysis of the PGP certificate graph. In *Proceedings of the New Security Paradigms Workshop 2002, Norfolk, VA*, September 2002.

[9] `http://www.umiacs.umd.edu/~mmarsh/CODEX/`.

[10] A. Datta, M. Hauswirth, and K. Aberer. Beyond "web of trust": Enabling P2P e-commerce. Technical Report IC/2003/06, Ecole Polytechnique Federale de Lausanne, 2003.

[11] Carl M. Ellison. The nature of a usable PKI. *Computer Networks*, 31(8):823–830, 1999.

[12] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI certificate theory, September 1999. RFC 2693.

[13] The Free Software Foundation. The gnu privacy handbook. `http://www.gnupg.org/gph/en/manual.html`.

[14] James M. Galvin. Public key distribution with secure DNS. In *Proceedings of the 6th USENIX Security Symposium*, pages 161–170, San Jose, CA, USA, July 1996. USENIX Association.

[15] Simson Garfinkel. *PGP: Pretty Good Privacy*. O'Reilly & Associates, 1994.

[16] Jennifer Golbeck and James Hendler. Accuracy of metrics for inferring trust and reputation. In *Proceedings of 14th International Conference on Knowledge Engineering and Knowledge Management*, Northamptonshire, UK, October 5–8 2004.

[17] J. P. Hubaux, L. Buttyan, and S. Capkun. The quest for security in mobile ad hoc networks. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHOC)*, October 2001.

[18] L.M. Kohnfelder. Towards a practical public-key cryptosystem, 1978. Undergraduate Thesis, MIT.

[19] Seungjoon Lee, Rob Sherwood, and Bobby Bhattacharjee. Cooperative peer groups in NICE. In *IEEE Infocom*, 2003.

[20] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *Proceedings of the twenty-first annual symposium on Principles of Distributed Computing*, pages 183–192. ACM Press, 2002.

[21] Sergio Marti, Prasanna Ganesan, and Hector Garcia-Molina. DHT routing using social links. In *IPTPS*, pages 100–111, 2004.

[22] Ruggero Morselli, Bobby Bhattacharjee, Michael A. Marsh, and Aravind Srinivasan. Efficient lookup on unstructured topologies. In *Symposium on Principles of Distributed Computing (PODC 2005)*, Las Vegas, Nevada, USA, July 2005. ACM SIGACT and SIGOPS.

[23] `http://www.openssl.org/`.

[24] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of the ACM SIGCOMM 2001 Technical Conference*, 2001.

[25] Michael K. Reiter, Matthew K. Franklin, John B. Lacy, and Rebecca N. Wright. The Ω key management service. *Journal of Computer Security*, 4(4):267–297, 1996.

[26] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 2001.

[27] Adam J. Slagell and Rafael Bonilla. PKI scalability issues. *ArXiv Computer Science e-prints*, September 2004.

[28] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the 2001 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '01)*, pages 149–160, San Diego, CA USA, August 27–31 2001. ACM.

[29] M. Drew Streib. Keyanalyze — analysis of a large OpenPGP ring. `http://dtype.org/keyanalyze/`. Data are from October 3, 2004.

[30] Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. COCA: A secure distributed online certification authority. *ACM Transactions on Computer Systems (TOCS)*, 20(4):329–368, 2002.

[31] P. Zimmerman. *The Official PGP User's Guide*. MIT Press, 1995.

A Overview of LMS

LMS is designed to work with any overlay network topology; links between nodes are specified by some external authority. For example, in our PKI these links will be determined by users expressing their trust of other users as certificate issuers. In LMS, data items are *virtualized* into some large identifier space with a distance metric. This virtualization should guarantee, with high probability, that identifiers are unique and

distributed uniformly (e.g., identifiers can be the SHA-1 hash of some data unique to the item). Each peer is also assigned a unique identifier from this space.

A fundamental parameter of the protocol is the neighborhood radius h: a peer p has knowledge of all peers within h hops of itself in the trust network, and we refer to these peers as p's *neighborhood*. Larger values of h increase the efficiency of the protocol but require storing more state at each peer. We assume $h = 2$ unless otherwise stated; thus, peers know their immediate neighbors and those neighbors' immediate neighbors. Optionally, p might limit the amount of state it needs to store by only keeping track of up to a certain number of neighbors, defined by a local parameter, and ignoring the rest. Obviously, this parameter must be at least as large as the set of p's immediate neighbors. We note, however, that the results of Section 6 show that the state required for each peer in the h-hop neighborhood is extremely small.

A peer p is a *local minimum* for an identifier x if there is no peer in p's neighborhood whose identifier is closer to x than p's identifier is. There will in general be many local minima for a given identifier, and LMS uses these minima as potential storage locations for the item with that identifier. When storing an item, the protocol selects some random subset of the minima at which to *replicate* the item. When searching for the item, LMS selects another random subset of the minima; if these subsets intersect the search is successful and a replica is returned.

LMS protocol operations send out *probes*, and any operation might send out multiple distinct probes (up to some configured maximum). Probe messages are initialized with an identifier x and a fixed random walk length ℓ. A node receiving the probe forwards it to a random neighbor until it has been forwarded ℓ times, at which point it is forwarded deterministically toward a local minimum for x. As shown in [22], this has the effect of forwarding the probe to a local minimum for x selected randomly from among all such minima in the network. Probes are used as follows:

- **Publishing.** A peer that wants to publish r replicas of an item with identifier x will initiate r probes requesting the eventual target (which will be some local minimum for x) to store a replica of x. The number r of replicas can be a preset parameter or it may be determined dynamically, such as by using an adaptive replication protocol [22].

- **Lookup.** A peer that wants to look up an item with identifier x initiates a sequence of up to s probes, hoping to find a local minimum for x which stores a replica of the item. Note that LMS is a *probabilistic* protocol, so an item may not be found even if it is currently stored in the system; however, this occurs with configurably small probability.

A detailed analysis of the LMS protocol appears in [22], but we recall the main result here. In a network of n peers and minimum neighborhood size d_h, the expected number of local minima for an item is $k = O(n/d_h)$. If r replicas of the item are placed and up to s search probes are sent, the probability that a search fails is $\exp(-\Omega(rs/k))$. Thus, if LMS places r replicas, a search requires $s = O(n/rd_h)$ probes in order to find a replica with constant probability. This translates into a search cost of $O(n \log n/rd_h)$, since it can be shown that it is sufficient to use $\ell = O(\log n)$ as the length of the random walk. For concreteness, in the interesting special case where $d_h = \Omega(n^{\frac{1}{3}})$ (i.e., the given topology has h-hop neighborhoods that are not too small) and where we choose r, s of the same order of magnitude, the search cost is $O(n^{1/3} \log n)$ and the number of replicas is $O(n^{1/3})$.

B Attacks Against the Web of Trust

The PGP web of trust model is subject to an intrinsic limitation due to the nature of trust. Suppose a malicious user A obtains a trust certificate from an honest user B. Then A can generate certificates for arbitrary identity-key mappings and provide a certificate chain for those mappings from any user that has a certificate chain to B. This may cause a honest user to accept incorrect information. How resilient the PKI

is to this attack depends on the trust aggregating function (Section 2), but is independent of how keys and chains are found (and, therefore, of our specific solution).

We argue that the problem described above is not an artifact of our choice of the web of trust model, but indeed applies to *any* PKI without a central authority. This is because, in the absence of CAs, the only source of authority is out-of-band information that principals have on the trustworthiness of other principals. If a malicious principal can subvert this out-of-band information and can wrongfully persuade an honest node of his trustworthiness, then the PKI will inevitably be affected.

Fundamentally, the weaknesses of the web of trust model stem from the fact that a web of trust is a social network, and thus is vulnerable to social engineering attacks. These sorts of attacks are common in the real world, and have often been used to bypass security systems. It is, in fact, possible to launch a social engineering attack on a centralized hierarchical CA, and such attacks can have considerably greater impact if undetected, since any certificate chain that traces correctly back to the root CA will be accepted by any principal trusting the root. In the web of trust model, social engineering attacks are more isolated by the gradual decrease of "trustedness" a node has as the distance to it increases.

C Generating Large PGP-Like Graphs

The degree distribution of both the unidirectional and bidirectional links in the web of trust graph conform to a power-law, i.e. $N(d) = \beta d^{-\alpha}$, where $N(d)$ is the number of nodes with degree d, for some $\beta > 0$ and $\alpha > 1$. We obtained log scale fits for all three (two unidirectional for incoming and outgoing edges and the bidirectional) sets of the α and β parameters. Let n_0, α_b and β_b be the number of nodes and the bidirectional degree distribution parameters in the PGP graph. For each new graph of size n, we generated a random undirected graph G'_n with a power-law degree distribution with parameters $\alpha = \alpha_b$ and $\beta = \beta_b n/n_0$. Next we generated a random directed power-law graph G''_n with similarly scaled parameters for the unidirectional outdegree and the indegree distributions. We finally defined our graph G_n by taking the union of the edges in G'_n and G''_n. In order to ensure strong connectivity, we added a random binary tree of bidirectional edges spanning the whole graph. In order not to inflate the number of edges, we then randomly removed $2n$ non-tree edges from the graph. In the graphs so generated, along with the power-law parameters, the average degree and the average neighborhood sizes were also preserved.